

**KLASIFIKASI MATA MENGANTUK MENGGUNAKAN EFFICIENTNETV2 +  
ATTENTION DAN DUAL-ATTENTION**

***COMPUTER VISION***

Diajukan untuk memenuhi nilai tugas proyek mata kuliah *Computer Vision*



Disusun oleh:

Nandito Yuda Samosir      215150301111014

Syauqi      215150307111010

PROGRAM STUDI TEKNIK KOMPUTER  
DEPARTEMEN TEKNIK INFORMATIKA  
FAKULTAS ILMU KOMPUTER  
UNIVERSITAS BRAWIJAYA  
MALANG  
2024

## PEMBAGIAN JOB DESK

NAMA	JOB DESK
Nandito Yuda Samosir (215150301111014)	<ul style="list-style-type: none"><li>- Melakukan riset dan memahami arsitektur EfficienNetV2 dan Attention</li><li>- Menghitung metrik evaluasi pada EfficienNetV2 dan Attention (akurasi, presisi, recall, F1-Score).</li><li>- Analisis confusion matrix dari hasil pengujian EfficienNetV2 dan Attention</li><li>- Analisis epoch terbaik pada EfficienNetV2 dan Attention</li></ul>
Syauqi (215150307111010)	<ul style="list-style-type: none"><li>- Melakukan riset dan memahami arsitektur EfficienNetV2 dan Dual Attention</li><li>- Menghitung metrik evaluasi pada EfficienNetV2 dan Dual Attention (akurasi, presisi, recall, F1-Score).</li><li>- Analisis confusion matrix dari hasil pengujian EfficienNetV2 dan Dual Attention</li><li>- Analisis epoch terbaik pada EfficienNetV2 dan Dual Attention</li></ul>

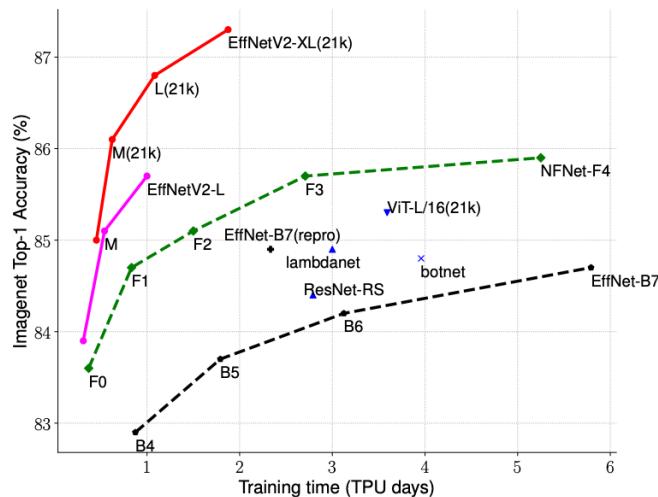
## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>3</b>
1. Metode yang Diusulkan.....	4
1.1. EfficientNetV2.....	4
1.2. EfficientNetV2 + Attention dan Dual-Attention.....	8
2. Pseudocode.....	12
2.1. EfficientnetV2.....	12
2.2. EfficientnetV2 + Attention.....	15
2.3. EfficientnetV2 + Attention dan dual-attention.....	17
3. Dataset Percobaan.....	22
4. Hasil.....	24
4.1. Epoch Terbaik EfficientNetV2.....	24
4.2. Epoch Terbaik EfficientNetV2 + Attention.....	25
4.3. Epoch Terbaik EfficientNetV2 + dual-attention.....	26
4.4. Confusion Matrix EfficientNetV2.....	28
4.5. Confusion Matrix EfficientNetV2 + Attention.....	31
4.6. Confusion Matrix EfficientNetV2 + Dual-Attention.....	35
4.7. Perbandingan.....	39
5. Kesimpulan.....	40
Ulasan Persamaan dan Perbedaan Penelitian.....	40
<b>REFERENSI.....</b>	<b>44</b>

## 1. Metode yang Diusulkan

### 1.1. EfficientNetV2

EfficientNetV2 merupakan salah satu keluarga dari jaringan konvolusi atau *Convolution Neural Network* (CNN) yang memiliki kecepatan pelatihan lebih cepat dan efisiensi parameter yang lebih baik daripada model sebelumnya. Pengembangan model ini menggunakan kombinasi *training-aware neural architecture search and scaling*, untuk mengoptimalkan kecepatan pelatihan dan efisiensi parameter secara bersamaan. Model dicari dari ruang pencarian yang diperkaya dengan operasi baru seperti *Fused-MBConv*. Eksperimen yang dilakukan pada penelitian terdahulu menunjukkan bahwa model EfficientNetV2 dilatih jauh lebih cepat daripada model canggih lain, sementara ukurannya hingga 66,8 kali lebih kecil.



(a) Training efficiency.

	EfficientNet (2019)	ResNet-RS (2021)	DeiT/ViT (2021)	EfficientNetV2 (ours)
Top-1 Acc.	84.3%	84.0%	83.1%	83.9%
Parameters	43M	164M	86M	24M

(b) Parameter efficiency.

**Gambar 1.1** ImageNet ILSVRC2012 top-1 Akurasi vs. Waktu Pelatihan dan Parameter

Sumber Tan, M., & Le, Q. (2021)

Arsitektur EfficientNetV2 merupakan pengembangan dari EfficientNet yang dirancang untuk meningkatkan efisiensi dan performa dalam tugas-tugas pembelajaran mesin. Komponen utama arsitektur EfficientNetV2:

1. *Fused-MBConv Layers*
2. *Mobile Inverted Bottleneck Convolution (MBConv)*
3. *Progressive Learning*

4. *Compound Scaling*
5. Optimasi Arsitektur dengan *Neural Architecture Search* (NAS)
6. Aktivasi Swish (SiLU)
7. *Squeeze-and-Excitation (SE) Modules*

*Progressive Learning* adalah pendekatan dalam pembelajaran mesin yang memungkinkan model untuk belajar secara bertahap dari data baru tanpa melupakan informasi yang telah dipelajari sebelumnya. Pada EfficientNetV2, *progressive learning* yang ditingkatkan digunakan untuk mempercepat pelatihan tanpa mengorbankan akurasi. Ini dilakukan dengan menyesuaikan regulasi secara adaptif seiring dengan peningkatan ukuran gambar selama pelatihan.

EfficientNetV2 memiliki beberapa varian yang berbeda, masing masing dirancang untuk berbagai kebutuhan dan ukuran model. Berikut beberapa jenis EfficientNetV2 yang umum digunakan:

1. EfficientNetV2-S, merupakan varian terkecil (*Small*) yang cocok digunakan untuk perangkat dengan sumber daya terbatas.
2. EfficientNetV2-M, merupakan varian menengah (*Medium*) yang menawarkan keseimbangan antara kinerja dan ukuran model.
3. EfficientNetV2-L, merupakan varian terbesar (*Large*) yang dirancang untuk kinerja maksimal dengan ukuran model yang lebih besar.

Selain ketiga model tersebut, terdapat pula model EfficientNetV2-B0, EfficientNetV2-B1, dan EfficientNetV2-B3. Model-model tersebut memiliki perbedaan pada jumlah parameter, FLOPs, *Infer-time*, waktu pelatihan, ukuran, dan juga Top-1 Accuracy.

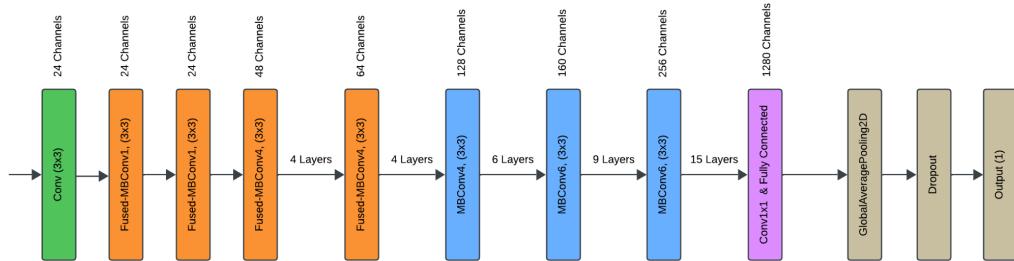
**Tabel 1.1** Perbandingan variasi model EfficientNetV2

Model	Top-1 Acc.	Parameter	Ukuran Input (px)
EfficientNetV2-B0	78.7%	7.1M	224x224
EfficientNetV2-B1	79.8%	8.1M	240x240
EfficientNetV2-B2	80.5%	10.1M	260x260
EfficientNetV2-B3	82.1%	14.4M	300x300
EfficientNetV2-S	83.9%	22M	384x384
EfficientNetV2-M	85.1%	54M	480x480
EfficientNetV2-L	85.7%	120M	480x480

Pada pengujian ini, variasi model EfficientNetV2 yang digunakan adalah EfficientNetV2-B0. Model ini memiliki lebih banyak parameter dan kompleksitas dibandingkan dengan variasi EfficientNetV2-B0 dan B1, tetapi masih lebih ringan dibanding model-B3. EfficientNetV2-M memiliki sekitar 8,7 juta parameter, yang berarti ukuran model ini lebih besar dibanding varian B0 dan B1, yang membuatnya lebih mampu menangkap fitur-fitur kompleks dari gambar. Ukuran gambar input yang digunakan adalah 260x260 piksel, yang lebih besar dibanding model sebelumnya. Ukuran input ini memungkinkan model untuk menangani detail yang lebih halus dalam gambar, seperti yang dilakukan pada penelitian ini.

Pemilihan varian EfficientNetV2-B2 sebagai model yang digunakan karena merupakan pilihan yang cukup baik untuk model yang membutuhkan akurasi cukup tinggi tanpa mengorbankan efisiensi komputasi. Dengan jumlah parameter yang cukup tinggi dan ukuran gambar yang lebih kecil mampu membuat pelatihan model lebih efisien dan cepat tanpa mengorbankan akurasi. Model ini cocok digunakan untuk tugas visi komputer yang memerlukan keseimbangan antara akurasi dan efisiensi.

### Arsitektur EfficientNetV2



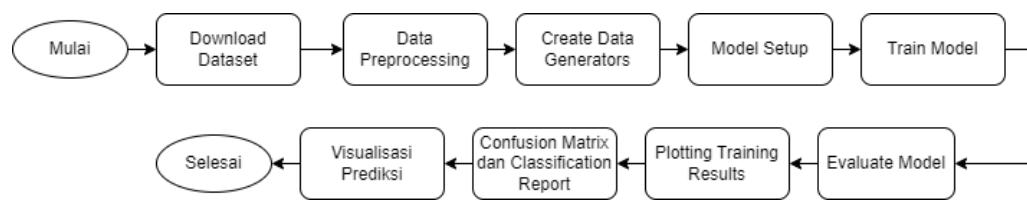
**Gambar 1.2** Arsitektur EfficientNetV2

- Lapisan Conv (3x3), merupakan lapisan awal dimana gambar input diproses menggunakan operasi konvolusi dengan filter 3x3. Tujuannya untuk menghasilkan representasi fitur awal dari gambar yang akan digunakan oleh lapisan-lapisan berikutnya.
- Lapisan *Fused-MBConv1* (3x3), merupakan versi efisien dari MBConv. Pada *Fused-MBConv*, langkah separasi *depthwise convolution* dan *pointwise convolution* digabung menjadi satu operasi. Ini mempercepat komputasi sambil tetap mempertahankan informasi penting dari input.
- Lapisan *Fused-EMBConv4* dan *EMBConv6* (3x3), adalah *Inverted Bottleneck Block* (MBConv), blok konvolusi yang telah dioptimalkan untuk bekerja dengan data yang lebih terkompresi. MBConv melibatkan ekspansi, *depthwise convolution*, dan reduksi dimensi. Digunakan untuk

menangkap fitur yang lebih kompleks dan detail pada tahap selanjutnya dari jaringan.

- *Global Average Pooling*, Setelah MBConv, dilakukan *pooling global* untuk mereduksi dimensi ruang sehingga menjadi satu dimensi untuk setiap *channel*. Ini meringkas informasi spasial dalam *feature map*.
- *Dense Layer* dan *Output Layer*, setelah *pooling*, lapisan ini melakukan klasifikasi akhir dengan menerapkan lapisan *fully connected (dense)* sebelum menghasilkan output akhir. Lapisan terakhir menghasilkan output yang sesuai dengan jumlah kelas atau output yang diminta, seperti klasifikasi objek.

### Flowchart



Gambar 1.3 Flowchart

1. **Mulai:** Titik awal proses, di mana seluruh pipeline dijalankan.
2. **Download Dataset:** Mengunduh dataset dalam bentuk file ZIP dari sumber, kemudian mengekstraknya untuk digunakan.
3. **Data Preprocessing:** Memproses data gambar termasuk augmentasi, normalisasi, dan mempersiapkan dataset agar siap digunakan untuk model training.
4. **Create Data Generators:** Membuat generator untuk memuat batch data secara efisien ke dalam memori saat training, validasi, dan pengujian.
5. **Model Setup:** Menyiapkan model arsitektur, dalam hal ini menggunakan EfficientNetV2, dan menambahkan lapisan-lapisan tambahan seperti GlobalAveragePooling, Dropout, dan Dense untuk klasifikasi.
6. **Train Model:** Melatih model pada dataset yang telah disiapkan menggunakan generator dengan optimasi dan callback yang diatur (seperti EarlyStopping).
7. **Evaluate Model:** Mengevaluasi kinerja model terhadap data pengujian, menghitung akurasi, dan metrik lainnya.
8. **Plotting Training Results:** Memvisualisasikan hasil training, seperti grafik akurasi dan loss untuk proses pelatihan dan validasi.

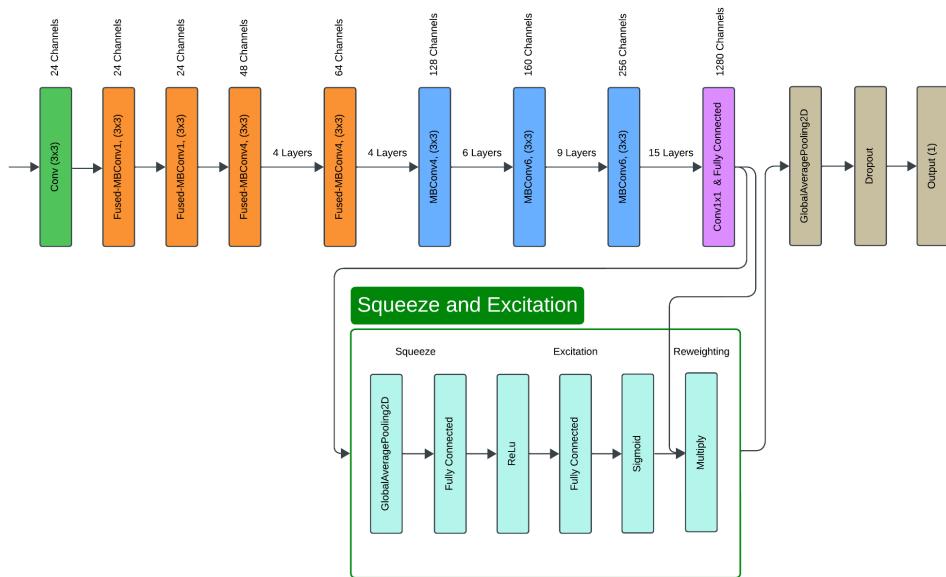
9. **Confusion Matrix dan Classification Report:** Menghasilkan confusion matrix serta laporan klasifikasi yang memberikan gambaran mengenai kinerja model pada data pengujian.
10. **Visualisasi Prediksi:** Menampilkan gambar hasil prediksi bersama dengan label asli untuk memeriksa performa model secara visual.
11. **Selesai:** Proses berakhir setelah semua evaluasi dan visualisasi selesai dilakukan.

## 1.2. EfficientNetV2 + Attention dan Dual-Attention

### 1.2.1. Single Attention (Squeeze-and-Excitation - SE)

Dalam perkembangan *Convolutional Neural Network* (CNN), salah satu fokus utama adalah meningkatkan kemampuan model dalam menangkap informasi yang relevan dari data input. Mekanisme *attention* diperkenalkan sebagai cara untuk memperkuat fitur-fitur penting dan menekan fitur-fitur yang kurang relevan. Salah satu implementasi *attention* yang populer dalam CNN adalah *Squeeze-and-Excitation (SE) Block*, yang dikenal sebagai bentuk *Single Attention*.

SE Block adalah modul yang dirancang untuk meningkatkan representasi fitur dalam jaringan CNN dengan memodelkan ketergantungan antar *channel* (saluran) dalam fitur. Ide utama dari SE Block adalah untuk memungkinkan jaringan belajar seberapa penting setiap *channel* dan mengkalibrasi fitur-fitur tersebut secara adaptif. Dalam CNN tradisional, operasi konvolusi menerapkan filter yang sama ke seluruh ruang fitur tanpa mempertimbangkan pentingnya setiap channel secara individual. Hal ini dapat menyebabkan fitur penting tidak mendapatkan perhatian yang cukup. SE Block bertujuan untuk mengatasi masalah ini dengan memperkenalkan mekanisme *attention* pada level *channel*.



**Gambar 1.4** Arsitektur EfficientNetV2 + Attention (*Squeeze and Excitation*)

### Proses

SE *Block* terdiri dari tiga langkah utama:

1. *Squeeze* (Pemerasan), bertujuan untuk mengompres informasi spasial untuk menangkap konteks global dari fitur. Proses menggunakan *Global Average Pooling* (GAP) pada setiap channel secara individual untuk menghasilkan representasi vektor yang menggambarkan keseluruhan informasi dalam *channel* tersebut. Output berupa vektor satu dimensi dengan panjang yang sama dengan jumlah *channel*.
2. *Excitation* (Eksitasi), bertujuan untuk memodelkan interaksi dan ketergantungan antar *channel*. Vektor hasil dari *squeeze* dilewatkan melalui dua lapisan *fully connected* (FC). Lapisan pertama mengurangi jumlah neuron dengan faktor *r* untuk mengurangi kompleksitas dan mengintroduksi *bottleneck*. Lapisan kedua mengembalikan jumlah neuron ke nilai aslinya. Lapisan pertama menggunakan aktivasi ReLU untuk non-linearity dan lapisan kedua menggunakan sigmoid untuk menghasilkan bobot skala antara 0 dan 1, yang merepresentasikan pentingnya setiap *channel*. Hasil dari proses ini adalah vektor skala dengan panjang yang sama dengan jumlah *channel*, dimana setiap elemen menunjukkan pentingnya *channel* tersebut.
3. *Reweighting* (Pengalihan Bobot), bertujuan mengkalibrasi ulang fitur pada setiap *channel* berdasarkan koefisien skala yang telah

dipelajari. Vektor skala dari langkah *Excitation* dikalikan secara *element-wise* dengan fitur input asli. Operasi ini memperkuat fitur-fitur pada channel yang dianggap penting dan menekan yang kurang penting. Outputnya adalah fitur yang dikalibrasi dengan *attention* pada *channel-channel* yang signifikan.

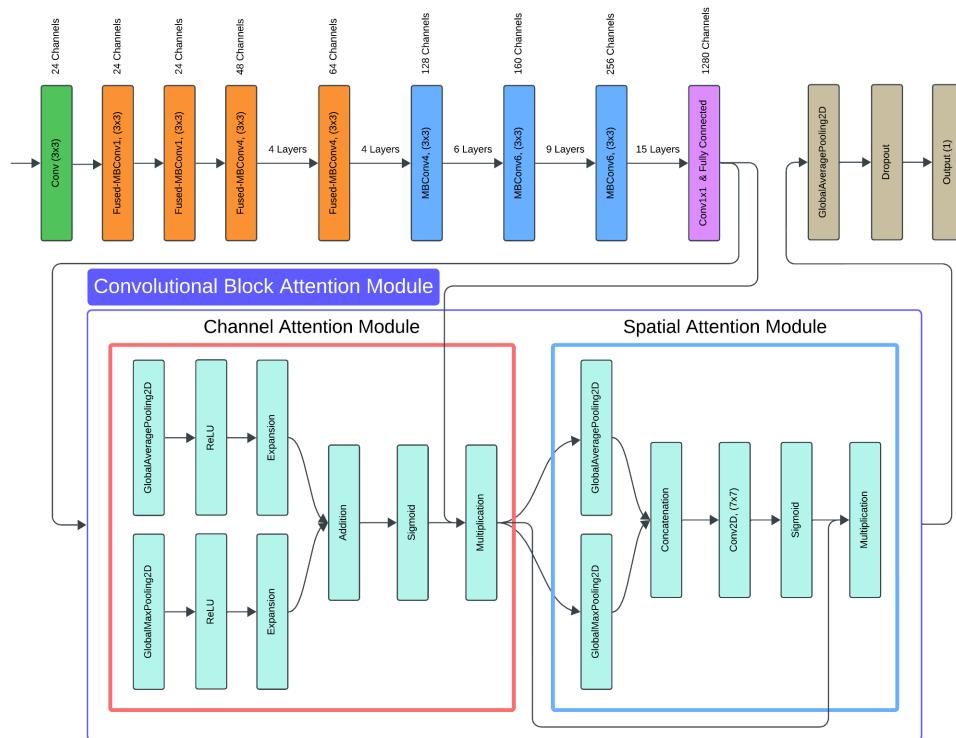
Proses SE *Block*:

1. *Input*: Fitur dengan dimensi (*Height, Weight, Channel*)
2. *Squeeze*: Melakukan *Global Average Pooling* untuk mengubah dimensi menjadi (1, 1, C), yang merepresentasikan konteks global tiap *channel*.
3. *Excitation*: Melalui dua lapisan FC dengan aktivasi ReLu dan sigmoid. Menghasilkan vektor skala (1, 1, C) yang merepresentasikan pentingnya masing-masing *channel*.
4. *Reweighting*: Mengalikan vektor skala dengan fitur input untuk mendapatkan fitur yang telah dikalibrasi.

#### 1.2.2. *Dual Attention (Convolutional Block Attention Module - CBAM)*

*Convolutional Block Attention Module* (CBAM) adalah mekanisme *dual attention* yang diperkenalkan untuk meningkatkan performa *Convolutional Neural Network* (CNN). CBAM memanfaatkan *Channel Attention* dan *Spatial Attention* secara berurutan untuk memodelkan apa dan dimana fitur penting berada dalam sebuah fitur. Tujuan utama dari CBAM adalah untuk meningkatkan representasi fitur dengan memfokuskan perhatian pada informasi yang lebih relevan secara adaptif.

Dalam jaringan CNN tradisional, semua fitur dianggap sama pentingnya, baik dari segi *channel* maupun spasial. Namun, pada kenyataannya, tidak semua fitur sama pentingnya untuk tugas tertentu. Dengan memberikan *attention* pada aspek *channel* dan spasial, model dapat lebih efektif dalam mengekstraksi informasi yang relevan dan mengabaikan yang tidak relevan. CBAM dirancang untuk dua hal utama, yaitu memperkuat fitur penting pada level *channel* (apa yang penting). Kemudian memperkuat lokasi penting pada level spasial (dimana bagian pentingnya berada).



**Gambar 1.4** Arsitektur EfficientNetV2 + Attention (*Squeeze and Excitation*)

CBAM terdiri dari dua modul utama yang diterapkan secara berurutan, yaitu *Channel Attention Module* dan *Spatial Attention Module*.

1. *Channel Attention Module*, bertujuan memodelkan hubungan antar *channel* untuk menentukan apa yang penting.

Proses:

- Input: Fitur  $F$  dengan dimensi  $(H, W, C)$
- *Global Pooling*: Menggunakan *Global Average Pooling* dan *Global Max Pooling* untuk menghasilkan dua vektor deskriptor channel.
- *Shared MLP (Multi-Layer Perceptron)*: Kedua vektor *pooling* dilewatkan melalui MLP bersama dengan satu *hidden layer*. Aktivasi ReLU digunakan pada *hidden layer*.
- Penggabungan: Keluaran dari kedua MLP digabungkan menggunakan *element-wise addition*.
- *Sigmoid Activation*: Menggunakan fungsi sigmoid untuk mendapatkan *attention map channel*  $M_C(F)$ .
- *Reweighting*: *Attention map* dikalikan dengan fitur input  $F$  secara *element-wise*.

Formula:

$$Mc(F) = \sigma(MLP(\text{AvgPool}(F)) + MLP(\text{MaxPool}(F)))$$

dimana  $\sigma$  adalah fungsi sigmoid.

2. *Spatial Attention Module*, bertujuan memodelkan hubungan spasial untuk menentukan dimana informasi penting berada.

Proses:

- Input: Fitur  $F'$  yang telah diperkuat secara *channel*.
- *Channel Pooling*: Menggunakan *Average Pooling* dan *Max Pooling* pada dimensi channel untuk menghasilkan dua peta fitur.
- *Concatenation*: Menggabungkan *Average Pooling Output* dan *Max Pooling Output* secara *channel-wise* untuk membentuk fitur dengan dua *channel*.
- *Convolutional Layer*: Melewaskan hasil konkatenasi melalui konvolusi dengan kernel  $7 \times 7$  untuk menghasilkan *attention map* spasial  $Ms(F')$ .
- *Sigmoid Activation*: menggunakan fungsi sigmoid untuk mendapatkan *attention map* spasial.
- *Reweighting*: *Attention map* dikalikan dengan fitur input secara *element-wise*.

Formula:

$$Ms(F') = \sigma(f^{7 \times 7}([\text{AvgPool}(F'); \text{MaxPool}(F')]))$$

di mana  $\sigma$  adalah fungsi sigmoid dan  $f^{7 \times 7}$  adalah operasi konvolusi dengan kernel  $7 \times 7$ .

## 2. Pseudocode

### 2.1. EfficientnetV2

```
# Import libraries
import necessary libraries (os, zipfile, shutil, tensorflow,
ImageDataGenerator, EfficientNetV2B2, preprocess_input, etc.)

# Download dataset from Google Drive
define URL for dataset
download dataset using gdown

# Extract dataset
define local path for zip file
extract the zip file into a specified directory

# Define directories for train, validation, and test data
```

```

define train_dir, val_dir, test_dir

# Set model parameters
set batch_size, img_height, img_width, epochs

# Create ImageDataGenerator for training data without
augmentation
train_datagen = ImageDataGenerator(preprocess_input)

# Create ImageDataGenerator for validation and test data without
augmentation
val_datagen = ImageDataGenerator(preprocess_input)
test_datagen = ImageDataGenerator(preprocess_input)

# Create generators for training, validation, and test data
train_generator = train_datagen.flow_from_directory(train_dir,
target_size, batch_size, class_mode='binary')
val_generator = val_datagen.flow_from_directory(val_dir,
target_size, batch_size, class_mode='binary')
test_generator = test_datagen.flow_from_directory(test_dir,
target_size, batch_size, class_mode='binary', shuffle=False)

# Load EfficientNetV2 model without top layers
base_model = EfficientNetV2B2(weights='imagenet',
include_top=False, input_shape=(img_height, img_width, 3))

# Freeze base model layers
base_model.trainable = False

# Add top layers (GlobalAveragePooling2D, Dropout, Dense)
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.2)(x)
outputs = Dense(1, activation='sigmoid')(x)

# Create the complete model
model = Model(inputs=base_model.input, outputs=outputs)

# Print model summary
model.summary()

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
loss='binary_crossentropy', metrics=['accuracy'])

# Add EarlyStopping callback
early_stop = EarlyStopping(monitor='val_accuracy', patience=6,
restore_best_weights=True)

# Measure training time
start_time_training = record current time

# Train the model
history = model.fit(train_generator, steps_per_epoch,
validation_data=val_generator, validation_steps, epochs)

```

```

end_time_training = record current time
training_time = end_time_training - start_time_training
print training time

# Measure testing time
start_time_testing = record current time

# Evaluate model on test data
test_loss, test_acc = model.evaluate(test_generator, steps)
print test accuracy

end_time_testing = record current time
testing_time = end_time_testing - start_time_testing
print testing time

# Plot training and validation accuracy
plot accuracy for both training and validation over epochs

# Plot training and validation loss
plot loss for both training and validation over epochs

# Predict on test data and create confusion matrix
y_pred = model.predict(test_generator)
y_pred = convert predictions to binary (0 or 1)
y_true = extract true labels from test_generator

create confusion matrix using confusion_matrix(y_true, y_pred)
display confusion matrix with seaborn heatmap

# Calculate accuracy, precision, recall, and F1 score
calculate accuracy, precision, recall, and F1 score
print the results

# Calculate average precision score
y_pred_proba = model.predict(test_generator)
average_precision = average_precision_score(y_true, y_pred_proba)
print average precision score

# Save model to file
model.save('efficientnetv2_drowsiness_model.h5')

# Download saved model file
download model file using Colab's files.download

# Generate classification report
classification_report = calculate and print classification report
based on y_true and y_pred

# Display random test images with predictions
define number of images, frame size, and figure layout

loop over num_images:
    randomly select image from test_dir
    resize and preprocess image
    predict class using the model
    display image and prediction in the subplot

```

```
show the plot with tight layout
```

## 2.2. EfficientnetV2 + Attention

```
# Import libraries
import necessary libraries for deep learning, plotting, and
metrics

# Download dataset
define dataset URL
download dataset using gdown
extract dataset from the downloaded zip file
check extracted directories

# Set directories for training, validation, and testing data
train_dir = path to train data
val_dir = path to validation data
test_dir = path to test data

# Define training parameters
batch_size = 32
img_height = 260
img_width = 260
epochs = 20

# Create ImageDataGenerator without data augmentation for
training
initialize ImageDataGenerator without augmentations
apply preprocessing for EfficientNetV2

# Create ImageDataGenerator for validation and testing (without
augmentation)
initialize ImageDataGenerator with only preprocessing for
EfficientNetV2

# Create generators for loading data
train_generator = load images from train_dir using
ImageDataGenerator
val_generator = load images from val_dir using
ImageDataGenerator
test_generator = load images from test_dir using
ImageDataGenerator, shuffle=False

# Define SE (Squeeze-and-Excitation) block
function se_block(input_tensor, ratio):
    apply global average pooling on input_tensor
    reshape pooled feature
    apply dense layer with reduced units (filters/ratio) and
    relu activation
```

```
    apply another dense layer with original filter size and
    sigmoid activation
    multiply input_tensor with the output of dense layers
    (attention mechanism)
    return reweighted feature

# Load base EfficientNetV2 model without top layers
base_model = load EfficientNetV2 with ImageNet weights, exclude
top layers

# Freeze layers of base model
set base model layers to non-trainable

# Add SE block and custom top layers
x = apply SE block on base model output
x = apply global average pooling
x = apply dropout
output = apply dense layer with sigmoid activation (for binary
classification)

# Create final model
model = create Model with base_model input and custom output

# Compile model
compile model with Adam optimizer, binary crossentropy loss, and
accuracy metric

# Define EarlyStopping callback to avoid overfitting
early_stop = EarlyStopping with patience and
restore_best_weights

# Start training timer
start timer for training

# Train the model
train the model using train_generator and val_generator with
defined epochs and early stopping

# End training timer and calculate training time
end timer for training

# Start testing timer
start timer for testing

# Evaluate model on test data
evaluate model using test_generator

# End testing timer and calculate testing time
end timer for testing

# Plot training and validation accuracy
plot accuracy curves for training and validation over epochs
```

```

# Plot training and validation loss
plot loss curves for training and validation over epochs

# Make predictions on test data
y_pred = model.predict on test_generator
convert predictions to binary labels (threshold = 0.5)

# Get true labels from test generator
y_true = extract true labels from test_generator
# Create and display confusion matrix
confusion_matrix = create confusion matrix from y_true and
y_pred
plot confusion matrix using seaborn heatmap

# Calculate metrics: accuracy, precision, recall, F1-score
calculate accuracy, precision, recall, and F1-score using
sklearn metrics
print these metrics

# Calculate average precision score
calculate average precision score from y_true and predicted
probabilities

# Save the trained model to file
save model as h5 file

# Generate classification report
classification_report = generate classification report from
y_true and y_pred
print classification report

# Display sample images with predictions
for each randomly chosen image from test_dir:
    load image, resize it, preprocess it
    make prediction using the model
    display image along with true label and predicted label
show images in a grid using matplotlib

```

### 2.3. EfficientnetV2 + *dual-attention*

```

# Import libraries
import necessary libraries (tensorflow, keras, gdown, etc.)

# Download dataset from Google Drive
url = 'download link'
download the dataset using gdown
extract dataset from zip file to a directory
# Set directories for train, validation, and test data
train_dir = 'path/to/train_data'

```

```

val_dir = 'path/to/val_data'
test_dir = 'path/to/test_data'

# Define parameters
batch_size = 32
img_height = 260
img_width = 260
epochs = 20

# Create ImageDataGenerators for train, validation, and test sets
train_datagen = ImageDataGenerator(preprocess_input)
val_datagen = ImageDataGenerator(preprocess_input)
test_datagen = ImageDataGenerator(preprocess_input)

# Create data generators for train, validation, and test sets
train_generator = train_datagen.flow_from_directory(train_dir,
batch_size, img_size, class_mode)
val_generator = val_datagen.flow_from_directory(val_dir,
batch_size, img_size, class_mode)
test_generator = test_datagen.flow_from_directory(test_dir,
batch_size, img_size, class_mode)

# Define CBAM block
function cbam_block(input_feature, ratio):
    # Channel attention mechanism
    apply average pooling and max pooling
    feed pooled features through fully connected layers
    combine features and apply sigmoid activation

    # Apply channel attention to input feature
    refine features based on channel attention

    # Spatial attention mechanism
    apply pooling to reduce spatial dimensions
    concatenate average and max pooling outputs
    apply convolution and sigmoid activation for spatial
    attention

    # Multiply input feature with spatial attention output
    return refined feature

# Load base EfficientNetV2 model (without top layers)
base_model = EfficientNetV2B2(weights='imagenet',
include_top=False, input_shape=(img_height, img_width, 3))

# Freeze base model layers
set base_model layers to non-trainable

# Add CBAM block on top of base model
x = base_model.output
x = cbam_block(x)
x = GlobalAveragePooling2D()(x)

```

```
x = Dropout(0.2)(x)
output_layer = Dense(1, activation='sigmoid')(x)

# Create complete model
model = Model(inputs=base_model.input, outputs=output_layer)

# Compile the model
compile model using Adam optimizer, binary crossentropy loss,
and accuracy metric

# Define EarlyStopping callback to prevent overfitting
early_stop = EarlyStopping(monitor='val_accuracy', patience=6,
restore_best_weights=True)

# Train the model
start training timer
history = model.fit(train_generator,
validation_data=val_generator, epochs=epochs,
callbacks=[early_stop])
end training timer and calculate total training time

# Evaluate the model on test data
start testing timer
test_loss, test_acc = model.evaluate(test_generator)
end testing timer and calculate total testing time

# Plot training and validation accuracy
plot accuracy vs epochs

# Plot training and validation loss
plot loss vs epochs

# Make predictions on test set
y_pred = model.predict(test_generator)
convert predictions to binary labels

# Get true labels from test data generator
y_true = test_generator.classes

# Generate and display confusion matrix
confusion_matrix = confusion_matrix(y_true, y_pred)
display confusion matrix using seaborn heatmap

# Calculate accuracy, precision, recall, F1-score
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)

# Print metrics
print accuracy, precision, recall, and F1-score

# Save the trained model to a file
```

```

model.save('model_filename.h5')

# Generate and print classification report
classification_report = classification_report(y_true, y_pred,
target_names=class_labels)
print classification report

# Display sample images with predictions
for i in range(num_images):
    load random image from test set
    preprocess image and predict its class using the model
    display image with true label and predicted label in a grid

# Show all images in grid
plt.show()

```

Pengaturan *Hyperparameter* yang digunakan dalam memperoleh model dapat dilihat pada tabel berikut:

**Tabel 2.1** Pengaturan *Hyperparameter* yang digunakan

<b>Hyperparameter</b>	<b>Nilai</b>	<b>Deskripsi</b>
Model	EfficientNetV2-B2	Model <i>backbone</i> yang digunakan tanpa lapisan atas (include_top=False)
<i>Batch Size</i>	32	Jumlah sampel per batch dalam proses <i>training</i>
<i>Image Height</i>	260	Tinggi input gambar
<i>Image Width</i>	260	Lebar input gambar
<i>Epochs</i>	20	Jumlah epoch maksimum dalam pelatihan
<i>Learning Rate</i>	0,001	Kecepatan pembelajaran yang digunakan dengan optimasi Adam
<i>Optimizer</i>	Adam	Optimizer yang digunakan untuk memperbarui bobot model
<i>Loss Function</i>	Binary Crossentropy	Fungsi loss yang digunakan untuk mengukur kesalahan dalam klasifikasi biner
<i>Early Stopping</i>	6 ( <i>patience</i> )	Menghentikan pelatihan jika akurasi validasi tidak meningkat dalam 6 epoch berturut turut

<i>Dropout Rate</i>	0,2	Rasio <i>dropout</i> pada lapisan <i>Global Average Pooling</i> untuk menghindari <i>overfitting</i>
<i>Image Preprocessing</i>	<code>preprocess_input</code>	Fungsi pre proses untuk skala input agar sesuai dengan kebutuhan EfficientNetV2

Berikut adalah gambaran garis besar tahapan deteksi dan pengolahan, khususnya terkait dengan penerapan model EfficientNetV2 beserta mekanisme Attention dan Dual Attention untuk mendeteksi mata mengantuk:

- Input Data, tahapan ini melibatkan pengumpulan dan persiapan dataset. Pada penelitian ini, data berupa gambar-gambar yang memerlukan deteksi kondisi tertentu, seperti mata mengantuk. Proses ini dimulai dari pengunduhan dataset, kemudian dilanjutkan dengan data preprocessing yang mencakup pemisahan dataset menjadi data latih, validasi, serta uji.
- Proses Data, pada tahapan ini akan menggunakan model EfficientNetV2 diimplementasikan sebagai *backbone* atau dasar arsitektur. Kemudian, model ini dikombinasikan dengan modul-modul attention (SE - Squeeze-and-Excitation) dan dual-attention (CBAM - Convolutional Block Attention Module) untuk memperkuat fitur penting dalam data gambar. EfficientNetV2 memiliki beberapa lapisan utama seperti:
  1. Fused-MBConv Layer dimana Lapisan ini mengkombinasikan depthwise dan pointwise convolution untuk memproses data secara efisien, mempercepat komputasi tanpa kehilangan informasi penting.
  2. Global Average Pooling: Menghasilkan representasi satu dimensi yang merangkum informasi spasial.
  3. Dense Layer: Mengelompokkan fitur yang telah dipelajari untuk menghasilkan output klasifikasi.

Kemudian menggunakan Attention Mechanism seperti :

1. Single Attention (SE Block), Mekanisme SE digunakan untuk meningkatkan relevansi fitur channel tertentu dengan memberikan perhatian pada fitur yang penting, dan mengabaikan fitur yang kurang relevan. SE melakukan proses squeeze (mengambil konteks global melalui pooling) dan excitation (mengalikan bobot yang telah dipelajari untuk meningkatkan fitur tertentu).

2. Dual Attention (CBAM), CBAM mengombinasikan perhatian secara spasial dan channel. Proses ini memastikan model tidak hanya mempertimbangkan relevansi antar channel tetapi juga lokasi spasial dalam gambar yang memuat informasi penting.

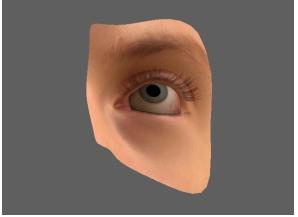
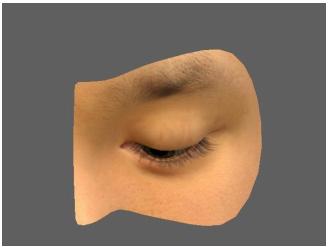
Setelah arsitektur model dan modul attention disusun, model dilatih menggunakan dataset yang telah disiapkan. Pelatihan ini melibatkan optimasi menggunakan algoritma backpropagation untuk memperbaiki bobot berdasarkan hasil prediksi model terhadap label sebenarnya. Pada tahap ini, beberapa teknik seperti early stopping atau callback lainnya digunakan untuk menghindari overfitting.

- Output, Evaluasi Model dan Visualisasi Hasil. Setelah pelatihan selesai, model dievaluasi menggunakan data uji. Pada laporan ini, evaluasi model melibatkan beberapa metrik seperti akurasi, confusion matrix, serta visualisasi prediksi melalui Grad-CAM untuk memberikan wawasan tentang area yang diperhatikan model saat membuat keputusan. Confusion Matrix Menyajikan performa klasifikasi model dalam bentuk tabel untuk setiap kategori yang diprediksi dengan benar dan salah. Grad-CAM Visualization: Menunjukkan visualisasi prediksi, di mana area yang paling diperhatikan oleh model diberi warna, membantu untuk memahami proses pengambilan keputusan yang dilakukan oleh model. Demikian gambaran garis besar tahapan input, proses penggabungan, dan output dari laporan computer vision terkait deteksi dan klasifikasi gambar.

### 3. Dataset Percobaan

Dataset percobaan terdiri dari dua kelas, yaitu “open” dan “closed”. Data percobaan secara keseluruhan memiliki jumlah 9869 gambar, dimana gambar tersebut dibagi lagi menjadi tiga bagian, yaitu data *train*, *validation*, dan *test*. Detail lengkap mengenai jumlah data tersebut dapat dilihat pada tabel berikut:

**Tabel 3.1** Dataset percobaan

Label	Train	Validation	Test	Total	Contoh
Open	3453	986	494	4933	
Closed	3455	987	494	4936	
Jumlah	6908	1973	988	9869	

Setelah dataset dibagi menjadi data *train*, *validation*, dan *test*, ukuran gambar dataset kemudian diubah sesuai dengan model EfficientNetV2-B2, yaitu 480x480 piksel. Salah satu alasan penggunaan EfficientNetV2-M dibanding model EfficientNetV2 yang lain adalah karena ukuran gambar input. Pada penelitian ini, model perlu melakukan deteksi dan lokalisasi dari objek yang membutuhkan detail yang cukup tinggi.

Pada penelitian ini, kode model dijalankan menggunakan *google colaboratory* (colab). Sistem tersebut memiliki spesifikasi dasar sebagai berikut:

1. *Operating System*: Linux 6.1.85+
2. CPU: Intel(R) Xeon(R) CPU @ 2.00GHz
3. RAM: 12.67 GB
4. GPU: Tesla T4
5. GPU Memory Total: 15360.0 MB
6. Python version: 3.10.12

Alasan penggunaan Google Colab pada penelitian ini karena akses penggunaan GPU yang mudah untuk membantu mempercepat waktu komputasi pelatihan model. Google Colab juga memungkinkan untuk berkolaborasi dengan tim secara mudah secara *real-time*.

## 4. Hasil

### 4.1. Epoch Terbaik EfficientNetV2

Pelatihan model dilakukan dengan 20 *epochs* dengan *early stop callbacks*. Pelatihan model dilakukan hingga *epochs* ke-8 karena *callbacks* terpicu. Epoch terbaik pada saat pelatihan terdapat pada epoch ke-6, dengan nilai *validation accuracy* mencapai 0,9524 dan *validation loss* sebesar 0,1429.

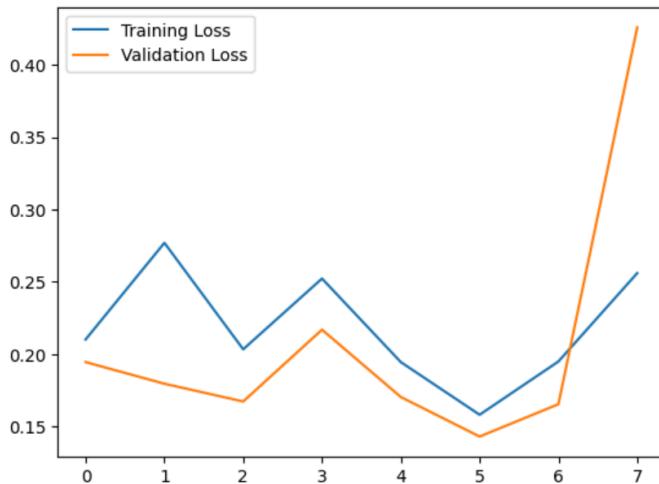
**Tabel 4.1** Epoch terbaik pada proses pelatihan model

Epoch 6/20	215/215	—	0s	301us/step	-
accuracy: 0.9688 -	loss: 0.1580 -	val_accuracy: 0.9524 -			
val_loss: 0.1429					



**Gambar 4.1** Grafik *train accuracy* dan *validation accuracy* setiap *epoch*

Berdasarkan grafik, epoch terbaik terlihat pada indeks ke-5, yang merujuk pada Epoch 6. Pada titik ini, model mencapai training accuracy tertinggi sekitar 0,97 dan validation accuracy sekitar 0,95 dengan validation loss yang terendah. Ini menunjukkan bahwa model memiliki keseimbangan terbaik antara performa pada data pelatihan dan validasi, tanpa tanda-tanda overfitting yang signifikan, menjadikan Epoch 6 sebagai yang terbaik untuk generalisasi model.



**Gambar 4.1** Grafik *train loss* dan *validation loss* setiap epoch

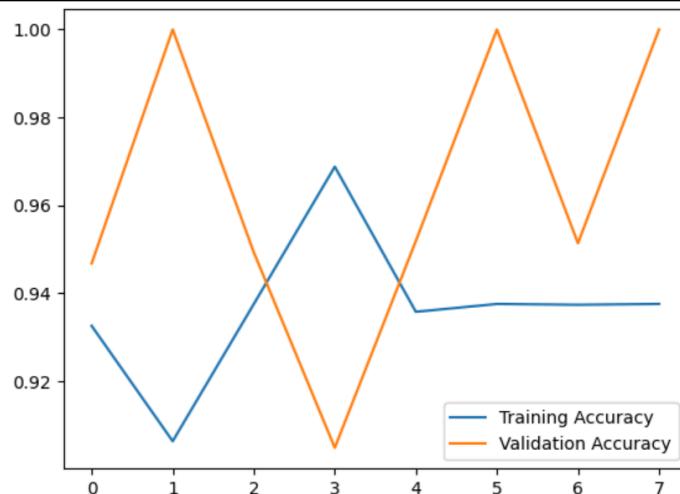
Grafik loss pada pelatihan model juga terlihat naik turun pada setiap epochnya. Nilai loss paling rendah terdapat pada epoch ke-6 (indeks ke-5), yaitu dengan nilai *validation loss* sebesar 0,1429.

#### 4.2. Epoch Terbaik EfficientNetV2 + Attention

Pelatihan model dilakukan dengan 20 *epochs* dengan *early stop callbacks*. Pelatihan model dilakukan hingga *epochs* ke-8 karena *callbacks* terpicu. Epoch terbaik pada saat pelatihan terdapat pada *epoch* ke-6, dengan nilai *validation loss* sebesar 0,0340.

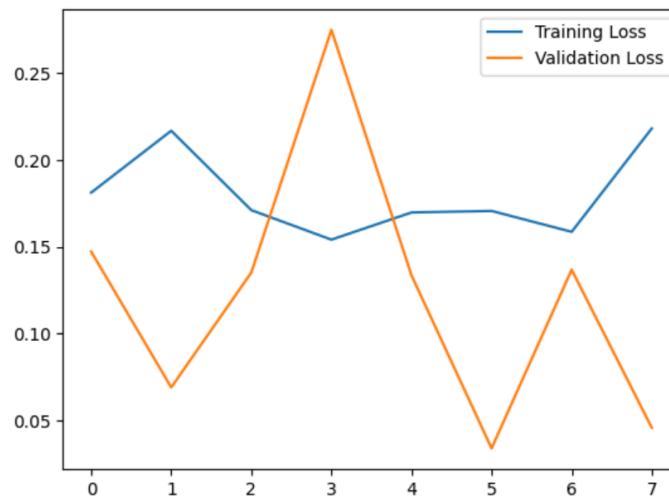
**Tabel 4.2** Epoch terbaik pada proses pelatihan model

Epoch 6/20	215/215	5s	21ms/step	-
accuracy: 0.9375	-	loss: 0.1706	-	val_accuracy: 1.0000
val_loss: 0.0340				



**Gambar 4.2** Grafik *train accuracy* dan *validation accuracy* setiap epoch

Berdasarkan grafik, epoch terbaik terlihat pada indeks ke-5, yang merujuk pada Epoch 6. Pada titik ini, model mencapai training accuracy tertinggi sekitar 0,94 dan validation accuracy sekitar 1,00 dengan validation loss terendah. Ini menunjukkan bahwa model memiliki keseimbangan terbaik antara performa pada data pelatihan dan validasi, tanpa tanda-tanda overfitting yang signifikan, menjadikan Epoch 6 sebagai yang terbaik untuk generalisasi model.



**Gambar 4.2** Grafik *train loss* dan *validation loss* setiap epoch

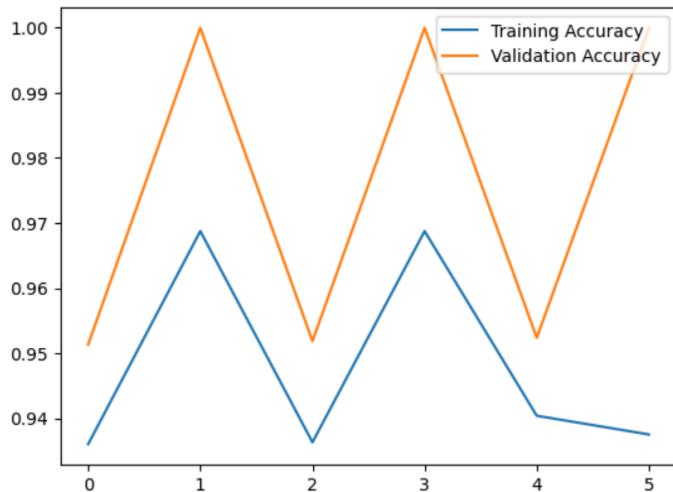
Grafik loss pada pelatihan model juga terlihat naik turun pada setiap epoch nya. Nilai loss paling rendah terdapat pada epoch ke-6, yaitu dengan nilai *validation loss* sebesar 0,0340.

#### 4.3. Epoch Terbaik EfficientNetV2 + *dual-attention*

Pelatihan model dilakukan dengan 20 *epochs* dengan *early stop callbacks*. Pelatihan model dilakukan hingga *epochs* ke-6 karena *callbacks* terpicu. *Epoch* terbaik pada saat pelatihan terdapat pada *epoch* ke-4, dengan nilai *validation loss* sebesar 0,0201.

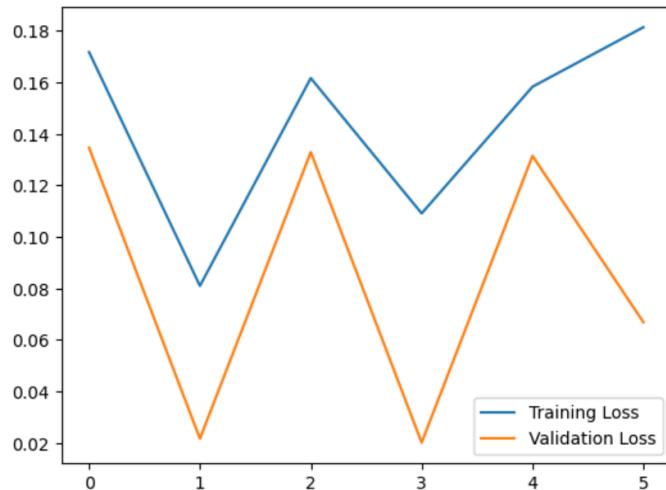
**Tabel 4.3** Epoch terbaik pada proses pelatihan model

Epoch 4/20	215/215	0s	396us/step	-
accuracy:	0.9688	-	loss:	0.1091 - val_accuracy: 1.0000 -
val_loss:	0.0201			



**Gambar 4.3** Grafik *train accuracy* dan *validation accuracy* setiap epoch

Berdasarkan grafik, epoch terbaik terlihat pada indeks ke-3, yang merujuk pada Epoch 4. Pada titik ini, model mencapai training accuracy sekitar 0,97 dan validation accuracy 1,00 dengan validation loss terendah. Ini menunjukkan bahwa model memiliki keseimbangan terbaik antara performa pada data pelatihan dan validasi, tanpa tanda-tanda overfitting yang signifikan, menjadikan Epoch 4 sebagai yang terbaik untuk generalisasi model.



**Gambar 4.3** Grafik *train loss* dan *validation loss* setiap epoch

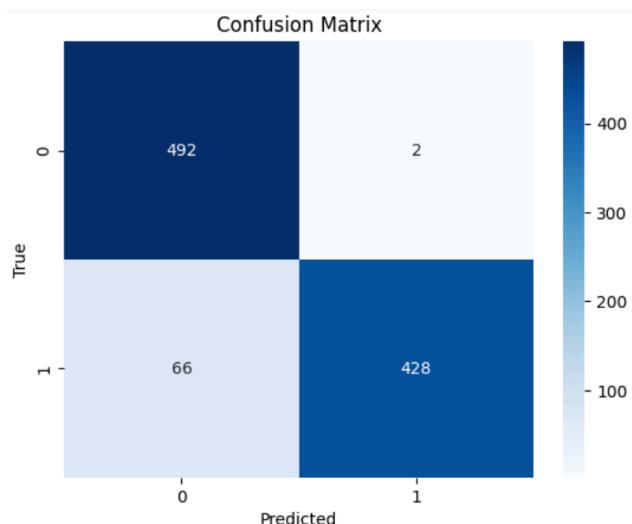
Grafik loss pada pelatihan model juga terlihat naik turun pada setiap epochnya. Nilai loss paling rendah terdapat pada epoch ke-4 (indeks ke-3), yaitu dengan nilai *validation loss* sebesar 0,0201.

#### 4.4. Confusion Matrix EfficientNetV2

Pada *confusion matrix*, terdapat 988 gambar yang diuji berdasarkan pembagian gambar dataset pada bagian *test*. Model memprediksi kelas *True negative* dengan jumlah 492. Ini berarti model berhasil memprediksi dengan benar 492 gambar kelas “closed” yang diuji. Model memprediksi kelas *False positive* dengan jumlah 2. Berarti model salah memprediksi kelas “open” menjadi “closed”. Tingkat kesalahan model masih cukup rendah dan dapat dikatakan masih dalam kondisi wajar.

Model memprediksi kelas *False negative*, dimana model memprediksi kelas “closed” dengan “open” sebanyak 66 kali dalam pengujian. Tingkat kesalahan model masih cukup rendah dan dapat dikatakan masih dalam kondisi wajar. Kemudian model memprediksi kelas *True positive* sebanyak 428 gambar. Dimana gambar dengan kelas “open” berhasil dengan tepat diprediksi dengan kelas “open” pula.

Secara keseluruhan, model menunjukkan performa yang baik, meskipun masih ada ruang untuk perbaikan dalam mendeteksi *False negative* dan mengurangi prediksi *False positive*.



**Gambar 4.4** Confusion Matrix EfficientNetV2

Closed	Open
$Precision = \frac{TN}{TN + FN}$	$Precision = \frac{TP}{TP + FP}$
$Precision = \frac{492}{492 + 66}$	$Precision = \frac{428}{428 + 2}$
$Precision = 0,8817$	$Precision = 0,9953$
$Recall = \frac{TN}{TN + FP}$	$Recall = \frac{TP}{TP + FN}$
$Recall = \frac{492}{492 + 2}$	$Recall = \frac{428}{428 + 66}$
$Recall = 0,9959$	$Recall = 0,8663$
$F1 Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$	$F1 Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$
$F1 Score = 2 \times \frac{0,8817 \times 0,9959}{0,8817 + 0,9959}$	$F1 Score = 2 \times \frac{0,9953 \times 0,8663}{0,9953 + 0,8663}$
$F1 Score = 0,9353$	$F1 Score = 0,9264$

**Tabel 4.4** Precision, Recall, F1-Score, Support EfficientNetV2

	Precision	Recall	F1-score	Support
Closed	0.88	1.00	0.94	494
Open	1.00	0.87	0.93	494
Accuracy			0.93	988
Macro Avg	0.94	0.93	0.93	988
Weighted Avg	0.94	0.93	0.93	988

#### 4.4.1. Jumlah Parameter EfficientNetV2

Model yang digunakan merupakan EfficientNetV2-B2 dengan tidak mengikutsertakan lapisan *fully-connected* di bagian atas. Model hanya akan menyertakan bagian konvolusional dari jaringan.

**Tabel 4.5** Jumlah parameter model EfficientNetV2

	Parameter	Size
Total params	8.770.783	33,46 MB
Trainable params	1.409	5,50 KB
Non-trainable params	8.769.374	33,45 MB

Model ini memiliki jumlah parameter sebesar 8.770.783 dengan ukuran 33,46 MB. Kemudian, untuk parameter yang dapat dilatih berjumlah 1.409 parameter dengan ukuran 5,50 KB. Beserta parameter yang tidak dapat dilatih sejumlah 8.769.374 parameter dengan ukuran 33,45 MB.

#### 4.4.2. mAP (*mean Average Precision*) EfficientNetV2

Nilai dari mAP (*mean Average Precision*) sebesar 0,9869. Dimana nilai tersebut sangat dekat ke 1, yang menandakan model memiliki performa hampir sempurna. Nilai presisi dan *recall* yang konsisten tinggi berarti model dapat melakukan klasifikasi yang sangat baik, serta prediksi sangat andal diberbagai ambang batas keyakinan (*confidence thresholds*).

```
print(f"Average precision score: {average_precision:.4f}")
```

```
31/31 ━━━━━━━━━━ 2s 79ms/step  
Average precision score: 0.9869
```

**Gambar 4.4.2** Nilai *mean Average Precision* (mAP) EfficientnetV2

#### 4.4.3. Waktu Komputasi EfficientnetV2

##### - *Training*

Proses pelatihan dilakukan sebanyak 8 epochs. Waktu komputasi pada setiap epoch berkisar antara 120 hingga 140 detik. Total waktu komputasi untuk pelatihan hingga epoch ke-20 adalah 524,14 detik.

```
Epoch 8/20  
215/215 ━━━━━━━━━━ 0s 354us/step  
Waktu komputasi training: 524.14 detik
```

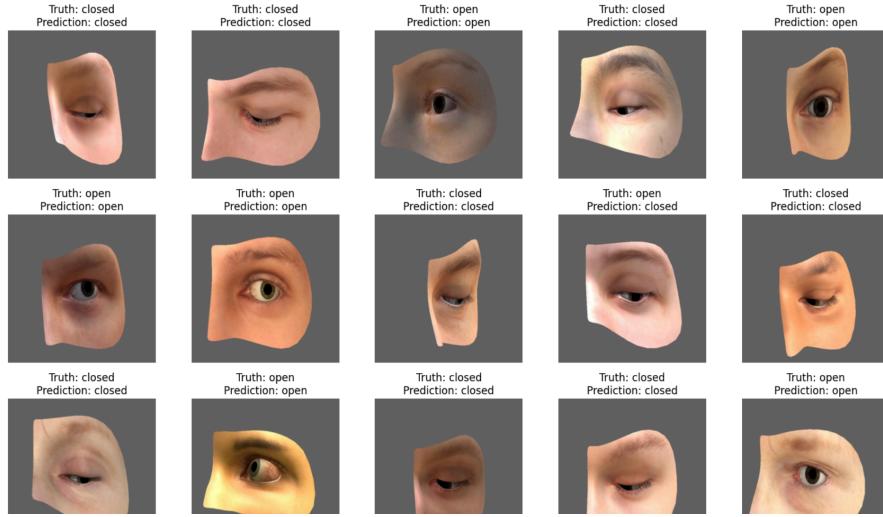
**Gambar 4.4.3** Waktu komputasi *training* EfficientnetV2

##### - *Test*

Proses pengujian dilakukan dengan mendapatkan prediksi pada Data uji. Total waktu yang dibutuhkan untuk melakukan pengujian pada data *test* adalah 2,78 detik.

```
30/30 ━━━━━━━━━━ 3s 84ms/step - accuracy: 0.9789 - loss: 0.1002  
Akurasi pada data testing: 0.934374988079071  
Waktu komputasi testing: 2.78 detik
```

**Gambar 4.4.3** Waktu komputasi *testing* EfficientnetV2



**Gambar 4.4.3** Contoh pengujian model EfficientnetV2

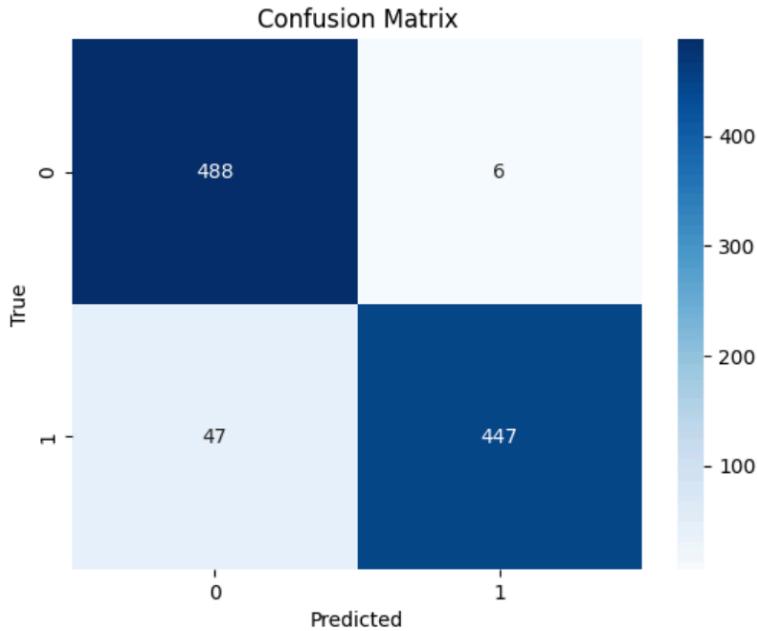
#### 4.5. *Confusion Matrix EfficientNetV2 + Attention*

Pada *confusion matrix*, terdapat 988 gambar yang diuji berdasarkan pembagian gambar dataset pada bagian *test*. Model memprediksi kelas *True negative* dengan jumlah 488. Ini berarti model berhasil memprediksi dengan benar 488 gambar kelas “*closed*” yang diuji. Model memprediksi kelas *False positive* dengan jumlah 6. Berarti model salah memprediksi kelas “*open*” menjadi “*closed*”. Tingkat kesalahan model masih cukup rendah dan dapat dikatakan masih dalam kondisi wajar.

Model memprediksi kelas *False negative*, dimana model memprediksi kelas “*closed*” dengan “*open*” sebanyak 47 kali dalam pengujian. Tingkat kesalahan model masih cukup rendah dan dapat dikatakan masih dalam kondisi wajar. Kemudian model memprediksi kelas *True positive* sebanyak 447 gambar. Dimana gambar dengan kelas “*open*” berhasil dengan tepat diprediksi dengan kelas “*open*” pula.

Berdasarkan hasil *confusion matrix*, penambahan model attention menunjukkan peningkatan dalam kemampuan memprediksi kelas “*open*” dengan benar, sebagaimana terlihat dari peningkatan jumlah *True positive* dari 428 menjadi 447. Namun, model attention juga meningkatkan jumlah kesalahan dalam memprediksi kelas “*open*” sebagai “*closed*” (*False positive* meningkat dari 2 menjadi 6). Selain itu, meskipun model attention lebih baik dalam mengurangi kesalahan memprediksi kelas “*closed*” sebagai “*open*” (*False negative* berkurang dari 66 menjadi 47), model ini sedikit lebih buruk dalam memprediksi kelas “*closed*” dengan benar, karena jumlah *True negative* turun dari 492 menjadi 488. Secara keseluruhan, penambahan *attention* memberikan perbaikan dalam

deteksi kelas "open", namun dengan sedikit pengorbanan pada prediksi kelas "closed", yang masih dapat ditingkatkan lebih lanjut.



**Gambar 4.5 Confusion Matrix EfficientNetV2 + Attention**

### Closed

$$Precision = \frac{TN}{TN + FN}$$

$$Precision = \frac{488}{488 + 47}$$

$$Precision = 0,9121$$

$$Recall = \frac{TN}{TN + FP}$$

$$Recall = \frac{488}{488 + 6}$$

$$Recall = 0,9878$$

$$F1 Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

$$F1 Score = 2 \times \frac{0,9121 \times 0,9878}{0,9121 + 0,9878}$$

$$F1 Score = 0,9336$$

### Open

$$Precision = \frac{TP}{TP + FP}$$

$$Precision = \frac{447}{447 + 6}$$

$$Precision = 0,9867$$

$$Recall = \frac{TP}{TP + FN}$$

$$Recall = \frac{447}{447 + 47}$$

$$Recall = 0,9048$$

$$F1 Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

$$F1 Score = 2 \times \frac{0,9867 \times 0,9048}{0,9867 + 0,9048}$$

$$F1 Score = 0,9440$$

**Tabel 4.6 Precision, Recall, F1-Score, Support EfficientNetV2 + Attention**

	Precision	Recall	F1-score	Support
Closed	0.91	0.99	0.95	494
Open	0.99	0.90	0.94	494
Accuracy			0.95	988
Macro Avg	0.95	0.95	0.95	988
Weighted Avg	0.95	0.95	0.95	988

#### 4.5.1. Jumlah Parameter EfficientNetV2 + Attention

Model yang digunakan merupakan EfficientNetV2-B2 dengan tidak mengikutsertakan lapisan *fully-connected* di bagian atas. Model hanya akan menyertakan bagian konvolusional dari jaringan. Kemudian, dilakukan penambahan model Attention yang dilakukan melalui *Squeeze-and-Excitation (SE)* Block. SE Block adalah jenis dari model attention yang berfokus pada penekanan atau peningkatan fitur tertentu berdasarkan pentingnya di saluran tertentu (*channel-wise attention*).

**Tabel 4.7 Jumlah parameter model EfficientNetV2 + Attention**

	Parameter	Size
Total params	9.020.087	34,41 MB
Trainable params	250.713	979,35 KB
Non-trainable params	8.769.374	33,45 MB

Model ini memiliki jumlah parameter sebesar 9.020.087 dengan ukuran 34,41 MB. Kemudian untuk parameter yang dapat dilatih berjumlah 250.713 parameter dengan ukuran 979,35 KB. Beserta parameter yang tidak dapat dilatih sejumlah 8.769.374 parameter dengan ukuran 33,45 MB.

#### 4.5.2. mAP (*mean Average Precision*) EfficientNetV2 + Attention

Nilai dari mAP (*mean Average Precision*) sebesar 0,9900. Dimana nilai tersebut sangat mendekati nilai 1, sehingga menandakan bahwa model ini memiliki performa yang hampir sempurna. Nilai presisi dan *recall* yang konsisten tinggi berarti model dapat melakukan klasifikasi yang sangat baik, serta prediksi sangat andal diberbagai ambang batas keyakinan (*confidence thresholds*).

```
print(f"Average precision score: {average_precision:.4f}")
```

```
31/31 ━━━━━━━━━━ 3s 93ms/step  
Average precision score: 0.9900
```

**Gambar 4.5.2** Nilai *mean Average Precision* (mAP) EfficientNetV2 + *Attention*

#### 4.5.3. Waktu Komputasi EfficientNetV2 + *Attention*

##### - *Training*

Proses pelatihan dilakukan sebanyak 8 epochs. Waktu komputasi pada setiap epoch berkisar antara 120 hingga 140 detik. Total waktu komputasi untuk pelatihan hingga epoch ke-8 adalah 477,63 detik.

```
Epoch 8/20  
215/215 ━━━━━━━━━━ 0s 292us/step  
Waktu komputasi training: 477.63 detik
```

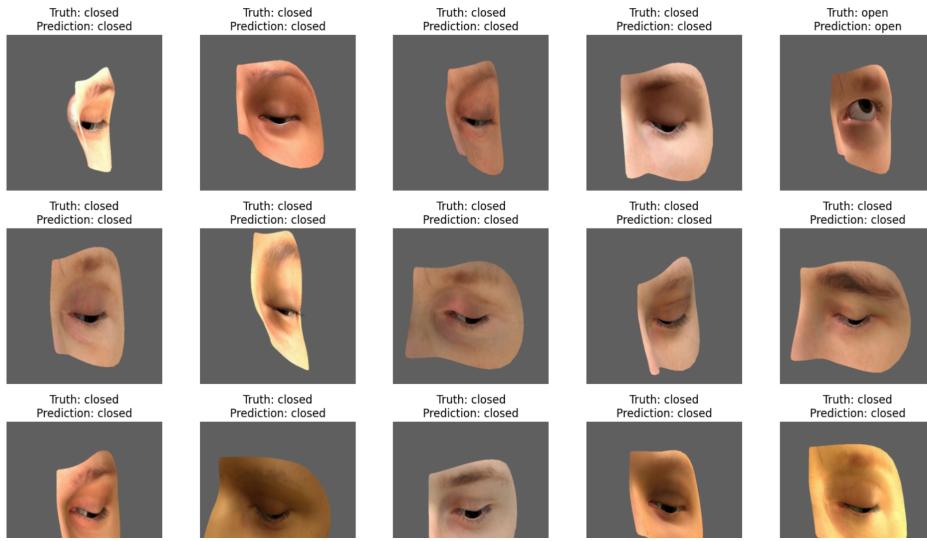
**Gambar 4.5.3** Waktu komputasi *training* EfficientnetV2 + *Attention*

##### - *Test*

Proses pengujian dilakukan dengan mendapatkan prediksi pada Data uji. Total waktu yang dibutuhkan untuk melakukan pengujian pada data *test* adalah 3,73 detik.

```
30/30 ━━━━━━━━━━ 4s 116ms/step - accuracy: 0.9757 - loss: 0.0882  
Akurasi pada data testing: 0.949999988079071  
Waktu komputasi testing: 3.73 detik
```

**Gambar 4.5.3** Waktu komputasi *testing* EfficientnetV2 + *Attention*



**Gambar 4.5.3** Contoh pengujian model EfficientnetV2 + Attention

#### 4.6. *Confusion Matrix EfficientNetV2 + Dual-Attention*

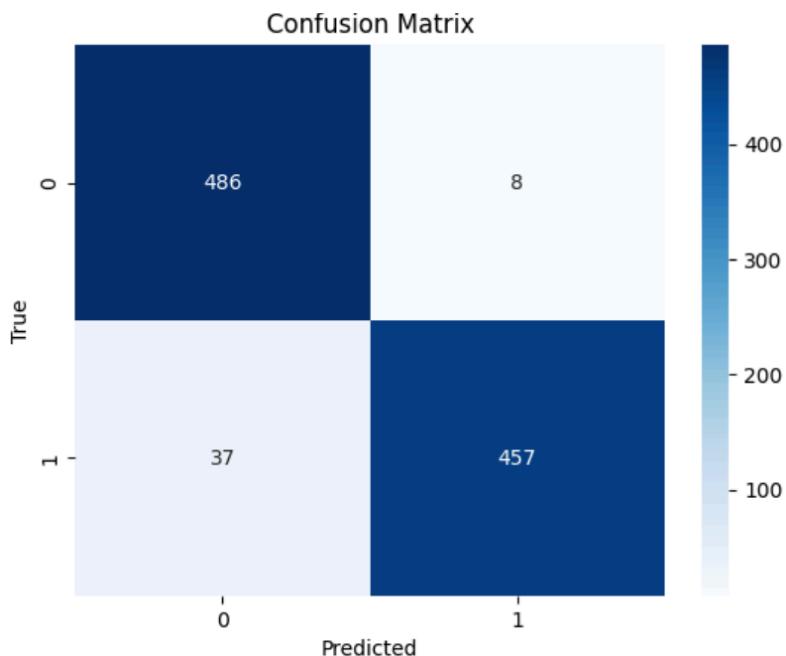
Pada *confusion matrix*, terdapat 988 gambar yang diuji berdasarkan pembagian gambar dataset pada bagian *test*. Model memprediksi kelas *True negative* dengan jumlah 486. Ini berarti model berhasil memprediksi dengan benar 486 gambar kelas "closed" yang diuji. Model memprediksi kelas *False positive* dengan jumlah 8. Berarti model salah memprediksi kelas "open" menjadi "closed". Tingkat kesalahan model masih cukup rendah dan dapat dikatakan masih dalam kondisi wajar.

Model memprediksi kelas *False negative*, dimana model memprediksi kelas "closed" dengan "open" sebanyak 37 kali dalam pengujian. Tingkat kesalahan model masih cukup rendah dan dapat dikatakan masih dalam kondisi wajar. Kemudian model memprediksi kelas *True positive* sebanyak 457 gambar. Dimana gambar dengan kelas "open" berhasil dengan tepat diprediksi dengan kelas "open" pula.

Berdasarkan hasil *confusion matrix*, dibandingkan dengan model EfficientNetV2 + attention, penambahan dual attention memberikan beberapa perubahan signifikan dalam performa prediksi. Model dual attention menunjukkan peningkatan dalam mengurangi jumlah *False negative*, dari 47 menjadi 37, yang berarti model lebih akurat dalam memprediksi gambar kelas "closed" sebagai "closed". Namun, dual attention meningkatkan jumlah *False positive* dari 6 menjadi 8, menandakan model ini sedikit lebih sering salah memprediksi gambar kelas "open" sebagai "closed." Selain itu, meskipun jumlah *True positive* meningkat dari 447 menjadi 457 pada dual attention, model ini

sedikit mengorbankan akurasi prediksi untuk kelas "closed," dengan penurunan True negative dari 488 menjadi 486.

Secara keseluruhan, dual attention memperbaiki deteksi pada kelas "open" dan mengurangi kesalahan pada prediksi kelas "closed" yang salah (False negative), tetapi ada sedikit kompromi dalam prediksi untuk kelas "open," dengan peningkatan False positive. Kedua model menunjukkan performa yang baik, namun dual attention memberikan sedikit peningkatan pada prediksi kelas "open" dengan pengorbanan kecil pada prediksi kelas "closed."



**Gambar 4.6** Confusion Matrix EfficientNetV2 + Attention dan Dual-Attention

### Closed

$$Precision = \frac{TN}{TN + FN}$$

$$Precision = \frac{486}{486 + 37}$$

$$Precision = 0,9292$$

$$Recall = \frac{TN}{TN + FP}$$

$$Recall = \frac{486}{486 + 8}$$

$$Recall = 0,9838$$

$$F1 Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

### Open

$$Precision = \frac{TP}{TP + FP}$$

$$Precision = \frac{457}{457 + 8}$$

$$Precision = 0,9827$$

$$Recall = \frac{TP}{TP + FN}$$

$$Recall = \frac{457}{457 + 37}$$

$$Recall = 0,9251$$

$$F1 Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

$$F1\ Score = 2 \times \frac{0,9292 \times 0,9838}{0,9292 + 0,9838} \quad F1\ Score = 2 \times \frac{0,9827 \times 0,9048}{0,9827 + 0,9048}$$

$$F1\ Score = 0,9585 \quad F1\ Score = 0,9531$$

**Tabel 4.8** Precision, Recall, F1-Score, Support EfficientNetV2 + Dual-Attention

	Precision	Recall	F1-score	Support
Closed	0.93	0.98	0.96	494
Open	0.98	0.93	0.95	494
Accuracy			0.95	988
Macro Avg	0.96	0.95	0.95	988
Weighted Avg	0.96	0.95	0.95	988

#### 4.6.1. Jumlah Parameter EfficientNetV2 + Dual-Attention

Model ini menggunakan arsitektur EfficientNetV2-B2 yang hanya menyertakan bagian konvolusional tanpa lapisan fully-connected di atasnya. Untuk meningkatkan performa deteksi objek, model dilengkapi dengan mekanisme dual attention, yang terdiri dari Channel Attention dan Spatial Attention.

**Tabel 4.9** Jumlah parameter model EfficientNetV2 + Dual-Attention

	Parameter	Size
Total params	9.268.081	35,35 MB
Trainable params	98.707	1,90 MB
Non-trainable params	8.769.374	33,45 MB

Model ini memiliki jumlah parameter sebesar 9.268.081 dengan ukuran 35,36 MB. Kemudian untuk parameter yang dapat dilatih berjumlah 498.707 parameter dengan ukuran 1,90 MB. Beserta parameter yang tidak dapat dilatih sejumlah 8.769.374 parameter dengan ukuran 33,45 MB.

#### 4.6.2. mAP (*mean Average Precision*) EfficientNetV2 + Dual-Attention

Nilai dari mAP (*mean Average Precision*) sebesar 0.9906. Dimana nilai tersebut sangat dekat ke 1, yang menandakan model memiliki performa hampir sempurna. Nilai presisi dan *recall* yang konsisten tinggi

berarti model dapat melakukan klasifikasi yang sangat baik, serta prediksi sangat andal diberbagai ambang batas keyakinan (*confidence thresholds*).

```
print(f"Average precision score: {average_precision:.4f}")

31/31 ━━━━━━━━━━ 3s 96ms/step
Average precision score: 0.9906
```

**Gambar 4.6.2** Nilai *mean Average Precision* (mAP) EfficientNetV2 +  
*Dual-Attention*

#### 4.6.3. Waktu Komputasi

##### - *Training*

Proses pelatihan dilakukan sebanyak 13 epochs. Waktu komputasi pada setiap epoch berkisar antara 120 hingga 140 detik. Total waktu komputasi untuk pelatihan hingga epoch ke-20 adalah 374,47 detik.

```
Epoch 6/20
215/215 ━━━━━━━━━━ 4s 20ms/step
Waktu komputasi training: 374.47 detik
```

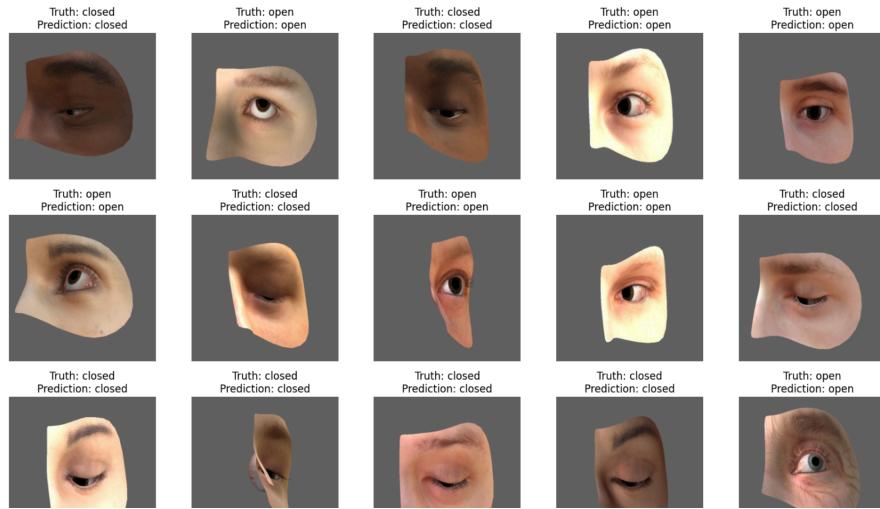
**Gambar 4.6.3** Waktu komputasi *training* EfficientnetV2 +  
*Dual-Attention*

##### - *Test*

Proses pengujian dilakukan dengan mendapatkan prediksi pada Data uji. Total waktu yang dibutuhkan untuk melakukan pengujian pada data *test* adalah 3,18 detik.

```
30/30 ━━━━━━━━━━ 3s 94ms/step - accuracy: 0.9735 - loss: 0.0868
Akurasi pada data testing: 0.9583333134651184
Waktu komputasi testing: 3.18 detik
```

**Gambar 4.6.3** Waktu komputasi *testing* EfficientnetV2 +  
*Dual-Attention*



**Gambar 4.6.3** Contoh pengujian model EfficientnetV2 + *Dual-Attention*

#### 4.7. Perbandingan

**Tabel 4.10** Perbandingan masing-masing

Keterangan	EfficientNetV2	EfficientNetV2 + <i>Attention</i>	EfficientNetV2 + <i>Dual-Attention</i>
Total Parameter	8.770.783	9.020.087	9.268.081
Akurasi Terbaik	0.9524	1.0000	1.0000
Loss (Val)	0.1429	0.0340	0.0201
mAP	0.9869	0.9900	0.9906
Waktu <i>Train</i> (avg)	130,50 s	115,25 s	123.00 s
Waktu <i>Test</i>	2,78 s	3,73 s	3,18 s
<i>Precision</i>	0,8663/0,9959	0,9878/ 0,9878	0,9292/0,9827
<i>Recall</i>	0,8663/0,9959	0,9878/0,9048	0,9838/0,9251
F1-Score	0,9353/0,9264	0,9336/0,9440	0,9838/0,9251

Berdasarkan **Tabel 4.10** di atas, nilai akurasi terbaik (*val accuracy*) dimiliki oleh model EfficientNetV2 + *Attention* dan EfficientNetV2 + *Dual-Attention*, dengan nilai akurasi validasi sama-sama sebesar 1,00. Nilai loss validasi terbaik dimiliki oleh EfficientNetV2 + *Dual-Attention* dengan nilai 0,0201. Nilai *Mean Average Precision* terbaik dimiliki oleh model EfficientNetV2 + *Dual-Attention* dengan nilai 0,9906. Berdasarkan nilai-nilai tersebut, dapat disimpulkan bahwa

model terbaik dimiliki oleh model EfficientNetV2 + *Dual-Attention*. Walaupun terdapat kekurangan dari model ini yaitu berupa jumlah parameter yang sedikit lebih besar daripada model lain, yang dapat menyebabkan ukuran file model menjadi sedikit lebih besar. Kemudian untuk rata-rata waktu pelatihan tiap *epoch* dari model bukan yang paling rendah, namun karena selisih rata-rata tersebut tidak terlalu signifikan sehingga bukan menjadi kekurangan mutlak. Waktu pengujian model juga bukan yang terbaik, namun sudah cukup baik. Total gambar yang digunakan pada pengujian adalah 988 gambar. Berarti waktu pengujian untuk satu gambar hanya membutuhkan 0,0032 detik.

## 5. Kesimpulan

Dengan akurasi dari pengujian model, dapat dikatakan penggunaan attention dan dual-attention dapat memberikan peningkatan kemampuan akurasi pengujian model. Penggunaan attention dan dual-attention juga memberikan peningkatan berbagai hal lain, seperti yang ditunjukkan pada tabel sebelumnya. Dengan demikian, dapat disimpulkan bahwa penggunaan attention dan dual-attention pada model EfficientNetV2 berhasil meningkatkan kemampuan dari model tersebut.

### Ulasan Persamaan dan Perbedaan Penelitian

No	Nama penulis (Tahun), Judul Penelitian	Persamaan	Perbedaan	Kelebihan	Kekurangan
1	Tan, M., & Le, Q. (2021), EfficientNetV2: Smaller Models and Faster Training	Menggunakan EfficientNetV2 sebagai <i>backbone</i> CNN dan fokus pada efisiensi parameter serta kecepatan pelatihan.	Penelitian referensi, mengembangkan model yang lebih ringan dan lebih cepat dengan teknik <i>progressive learning</i> , tidak berfokus pada <i>dual attention</i> .	Lebih mendalam pada fitur <i>channel</i> dan spasial melalui modul <i>attention</i> .	Jumlah parameter yang bertambah banyak karena penggunaan modul <i>attention</i> yang dapat mengakibatkan ukuran model lebih besar dan butuh biaya komputasi

					lebih tinggi pada penggunaan.
2	Hu, et al. (2018), Squeeze-and-Excitation Networks	Menggunakan <i>Squeeze-and-Excitation</i> (SE) untuk menonjolkan fitur penting dalam citra	Penelitian referensi, menggunakan <i>SE block</i> untuk memperkuat fitur kanal tanpa <i>dual attention</i>	Kombinasi SE dan CBAM dalam penelitian memberikan <i>attention</i> lebih baik pada fitur penting di area spesifik	Kompleksitas arsitektur lebih tinggi karena gabungan <i>dual attention</i> dan SE, menambah kebutuhan komputasi
3	Yang, et al. (2021), A Dual Attention Network Based on EfficientNet-B2 for Short-term Fish School Feeding Behavior Analysis in Aquaculture	Menggunakan EfficientNet dan <i>dual attention</i> untuk meningkatkan akurasi dalam klasifikasi citra.	Penelitian referensi fokus pada analisis perilaku makan ikan dalam industri perikanan, bukan pada deteksi mata mengantuk.	Fokus lebih spesifik pada deteksi mata mengantuk yang mungkin relevan diterapkan pada pengamatan kondisi pengemudi.	Kurang mempertimbangkan teknik pembobotan fitur dan latar belakang yang kompleks seperti pada citra <i>aquaculture</i> .
4	Woo, S., et al. (2018), Cbam: Convolutional block attention module	Menerapkan <i>dual attention</i> pada model CNN dengan menambahkan mekanisme <i>attention</i> yang fokus pada <i>channel</i> dan <i>spatial</i> .	CBAM divalidasi melalui eksperimen ekstensif pada dataset ImageNet01K, MS COCO, dan VOC 2007	Penelitian membandingkan model EfficientNetV2 normal, ditambah modul <i>single attention</i> , dan modul <i>dual attention</i> untuk memvalidasi perbedaan yang terjadi.	Data yang digunakan pada penelitian ini merupakan gambar yang dibuat dengan lingkungan terkontrol, sehingga kemungkinan akan memberikan performa yang berbeda

					apabila diaplikasikan pada dunia nyata.
5	Xiao, et al. (2023). Apple Leaf Pathology Detection Based on Improved EfficientNetV2	Menggunakan model EfficientNetV2 sebagai arsitektur utama dan menambahkan mekanisme attention untuk meningkatkan performa model dalam klasifikasi gambar. Selain itu, penelitian dilakukan untuk meningkatkan akurasi deteksi objek berdasarkan gambar, serta mengoptimalkan efisiensi dan ketepatan model.	Digunakan untuk deteksi penyakit pada daun apel dengan menggunakan Dilated Convolution dan CBAM untuk meningkatkan performa pada deteksi penyakit daun apel, yang membantu memperluas bidang perceptual tanpa memperbesar ukuran model.	Melakukan pelatihan sebanyak 200 epoch sehingga meningkatkan akurasi dan kinerja model pada data pelatihan yang mencapai 98%.	Membutuhkan waktu pelatihan yang lama dengan 200 epoch, yang menunjukkan bahwa model memerlukan sumber daya komputasi yang lebih besar dan waktu pelatihan yang cukup lama untuk mencapai performa optimal. Selain itu, ukuran model sedikit lebih besar (84.6 MB), yang meskipun masih ringan, bisa menjadi kendala untuk beberapa perangkat dengan keterbatasan memori atau daya tahan baterai, terutama pada

					perangkat embedded.
6	Zhang, L. et al. (2022), <i>Progressive Dual-Attention Residual Network for Salient Object Detection</i>	Menggunakan dual-attention untuk meningkatkan deteksi objek	Fokus pada deteksi objek salien secara umum, bukan pada deteksi mata mengantuk	Metode saliency map dapat meningkatkan perhatian model pada fitur penting dalam gambar	Metode ini dirancang untuk objek umum, mungkin kurang optimal untuk mendeksi detail spesifik seperti mata mengantuk
7	Tafala, I., et al. (2024), <i>EfficientNetV2 and Attention Mechanisms for the Automated Detection of Cephalometric Landmarks</i>	Menggunakan EfficientNetV2 dan attention untuk mendeksi landmark medis	Fokus pada landmark cephalometric dalam kedokteran gigi, bukan pada analisis perilaku mata	Memanfaatkan attention untuk menonjolkan landmark penting, meningkatkan akurasi deteksi	Dataset kecil dalam kedokteran gigi mungkin mengurangi kemampuan generalisasi ke tugas deteksi yang berbeda
8	Pacal, I., et al. (2024), <i>Enhancing EfficientNetv2 with Global and Efficient Channel Attention Mechanisms for Accurate MRI-Based Brain Tumor Classification</i>	Menggunakan EfficientNetV2 dengan mekanisme channel attention	Fokus pada klasifikasi tumor otak dari citra MRI, bukan pada deteksi mata mengantuk	Menggunakan attention untuk meningkatkan fitur salien pada citra MRI, mencapai akurasi tinggi	Dirancang untuk deteksi tumor yang mungkin kurang fleksibel untuk deteksi perilaku berbasis visual seperti mata mengantuk

## REFERENSI

- Tan, M., & Le, Q. (2021, July). Efficientnetv2: Smaller models and faster training. In International conference on machine learning (pp. 10096-10106). PMLR.
- Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7132-7141).
- Yang, L., Yu, H., Cheng, Y., Mei, S., Duan, Y., Li, D., & Chen, Y. (2021). A dual attention network based on efficientNet-B2 for short-term fish school feeding behavior analysis in aquaculture. Computers and Electronics in Agriculture, 187, 106316.
- Woo, S., Park, J., Lee, J. Y., & Kweon, I. S. (2018). Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 3-19).
- Guo, Y., Wang, Y., Yang, H., Zhang, J., & Sun, Q. (2022). Dual-attention EfficientNet based on multi-view feature fusion for cervical squamous intraepithelial lesions diagnosis. Biocybernetics and Biomedical Engineering, 42(2), 529-542.
- Xiao, H., Shen, H., Zhu, H., Ye, J., Deng, S., & Xiao, Z. (2023, April). Apple Leaf Pathology Detection Based on Improved EfficientNetV2. In *2023 IEEE 3rd International Conference on Electronic Communications, Internet of Things and Big Data (ICEIB)* (pp. 1-6). IEEE.
- Zhang, L., Zhang, Q., & Zhao, R. (2022). Progressive Dual-Attention Residual Network for Salient Object Detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(9), 5902-5913.
- Tafala, I., Ben-Bouazza, F.-E., Edder, A., Manchadi, O., Et-Taoussi, M., & Jioudi, B. (2024). EfficientNetV2 and Attention Mechanisms for the Automated Detection of Cephalometric Landmarks. In *2024 International Conference on Intelligent Systems and Computer Vision (ISCV)*.
- Pascal, I., Celik, O., Bayram, B., & Cunha, A. (2024). Enhancing EfficientNetv2 with Global and Efficient Channel Attention Mechanisms for Accurate MRI-Based Brain Tumor Classification. *Cluster Computing*, 27(11187-11212).